Deduction as an Engineering Science

Dieter Hutter

German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany

Abstract

Although in recent years considerable progress has been made in the theory of automated theorem proving, the use of theorem provers in practice is still more or less restricted to a limited number of academic groups. A lot of effort has been spent in techniques to optimize the underlying logic engine by, for instance, developing efficient datastructures or controlling redundancy in large search spaces (see [29]). However, the development of techniques and methodologies to integrate such a logic engine into an overall proof assistant has gained less attention. In this paper we discuss the related research problems and will explore possible ways to tackle these problems.

1 Introduction

Automated deduction has been an active area since the 1950s. However, the ultimate goal of fully mechanizing the proof capability of a mathematician is still distant. To overcome the combinatorial explosion in proof search, specialized theorem provers have been developed that are restricted to specific domains. Deduction engineering, which denotes the process of adjusting a functioning deductive system for improved performance, has become the source of many improvements in the area of automated deduction.

As Loveland states in [23], deduction engineering covers, for instance, the process of strategy formulations, having a range of outcomes, from publishable restrictions of considerable sophistication to simple delaying of the use of clauses that have many free variables. Most progress is found now by augmenting existing systems rather than implementing new basic procedures. Successful systems are composites and will become more so. Resources are needed to let the systems grow in capability, for instance, to allow for adding new heuristics or strategies.

[1] Email: hutter@dfki.de

Originally supposed to provide **the** reasoning capabilities for Artificial Intelligence applications, the focus for applying deduction has mainly shifted to the area of formal methods. To increase the reliabity of complex software systems, the use of formal software development and program verification in particular becomes more and more popular. While nowadays the use of formal methods is state of the art in industrial hardware design, still verification techniques have not penetrated software manufacturing industry but their utilization is restricted to a rather limited number of academic groups and only slowly starts to be applied in industry. It has been claimed [29] that automated theorem proving has made major improvements because of more efficient datastructures and algorithms and also because of the speedup in hardware. While this progress has certainly resulted in the automation of proofs for highly complex problems (e.g. [24]), the use of these systems is restricted to higly skilled and educated users. Still an overall methodology and corresponding techniques are missing to embed such systems into a more general proof engineering process. The more the problems are getting complex and the more the construction of proofs absorbs user's time, the more there is a need for a methodological, a technical and a tool support to conduct, to organize, and to support the time-consuming proof construction process. Analogously to the area of software engineering, there is definitely a need for a methodology that we call *proof engineering*.

In this paper we will discuss the arising issues of proof engineering that emerge when aiming at a more widespread application of deduction in software development.

2 Proof Engineering as an Evolutionary Process

Developing proofs in applications of formal methods is a lengthy and error-prone task, which cannot be fully automated in the near future. Even so-called push-button technologies, like model checking, require creative user interaction to guide the construction of appropriate problem abstractions [22] which prevent combinatorical explosion during the proof search. Some approaches to automate the generation of this abstraction even involve the use of a deduction system (e.g. [12]). The verification of small-sized industrial developments typically requires several person months establishing all arising proof obligations (see [15]). Since the conducted proofs are only valid relative to the given specification, errors of the specification revealed in late verification phases pose an incalculable risk for the overall project costs. However in all case studies, development steps turned out to be flawed and errors had to be corrected. The search for formally correct software and the corresponding proofs is more like a *formal reflection* of partial developments rather than just a way to assure and prove more or less evident facts.

Hence there is a need for techniques and corresponding tools to cope with the *evolutionary* nature of the development of formal proofs. Systems are

needed that allow for an incremental design and development, revision and repair of formal proofs. In the area of formal methods, techniques have been developed to allow for a management of change in structured formal developments [1,5,6,16]. The MAYA-system, for instance, maintains relations between different theories in structured specifications and tries to minimize the proof obligations that arise when formulating or changing specifications. Maybe in the future, the techniques developed in this area will result in an evolutionary formal software development.

Analogously to software engineering, we envisage a methodology for an evolutionary proof construction. Tools for automated theorem proving should support the entire construction process from the initial formalization of the problem to its final verification and that should gracefully adjust this process if the user changes the settings of the problem or slightly modifies proof obligations. Part of this methodology are techniques for the reuse of proofs. Various techniques have been proposed in the past (e.g. [20,7,21,26]). However, none of these techniques has been integrated into an overall proof management of an existing system. The challenging problem of how to reuse a proof of a simple problem to incrementally find solutions for more and more refined versions of the problem is (up to the author's knowledge) still an untackled research topic. Also providing support for the proof design in early phases is still unsolved. Using automated theorem provers successfully typically requires that the user knows already the key steps of the desired proof in advance. Depending on the way of specifying the initial problem, a theorem prover will be able to solve a problem or not. However, the knowledge about the *successful* way requires internal knowledge on the automation of proof search and is usually restricted to the developers or rather experienced users of the system. Evolving proofs by gradually proving more and more detailed versions and reusing the proofs of the previous versions might provide a suitable way to guide theorem provers without knowing the details of the search engine.

3 The Role of the User

Today, the tackling of proof obligations in formal methods requires user interactions and will do it in the foreseeable future. While in automated theorem proving user interaction is restricted to the fine-tuning of various parameters of a system (with mostly unpredictable results), tactical theorem proving requires the adequate selection of a series of tactic/tactical calls to find a proof. Failures of such a system to find a proof typically give rise to a laborious investigation of possible reasons and require far-reaching knowledge of the user how the underlying proof calculus and the proof procedure explores the search space. As a result, a user has already to know the key steps of a proof in advance to supervise the progress of the proof search. Up to now deduction systems do not provide abstract information about what they achieved, why they failed, or what type of information is missing to enable a more successful

proof search. In inductive theorem proving, the notion of critics [18] was a first step towards such an approach. In this area we have rather precise expectations of the outcome of a deduction process which allows us to evaluate the value of derived formulas.

There is a need for techniques to allow for a more abstract communication between systems and users about the achievements, the goals, and the intentions in the ongoing proof work. Prerequisites are the development of appropriate abstractions to provide a language for communication and the design of appropriate user interfaces that ease the communication.

According to [11], abstract proof search is a process by which, starting from a representation of a problem at a so-called ground level, we construct a new and simpler representation at a so-called abstract level and use it to solve the original problem. Different techniques to abstract from details have been studied in the literature. Examples of abstractions in theorem proving are, for instance, to collapse constants, functions or predicate symbols [13,28,11,10] or to drop arguments to function or predicate symbols [25]. The problem is how to find an appropriate solution. If we abstract too much information then we often obtain abstract solutions that cannot be transferred to the ground level. Then, planning at the abstract level is even more difficult than planning at the ground level because the abstraction removes necessary control information, or we obtain only little information from the abstract proof how to guide the proof at the ground level. On the other hand, if we abstract too little then the complexity of finding a proof at the abstract level is almost equal to the complexity at the ground level, which only shifts the search problem from the ground level to an equally hard one at the abstract level. In the past it turned out that in practice the type of abstractions, mentioned above, is of limited use [28]. Rippling can be seen as a first example of a successful use of abstractions in theorem proving. The reason for this success is that this abstraction incorporates additional knowledge on the application domain (inductive proofs).

4 Integrating Application Knowledge into a Prover

Proof obligations increase dramatically in size when tackling more complex (industrial scaled) verification problems. Typically these examples come along with a large set of axioms most of them irrelevant for the proof. However, most theorem proving systems cannot deal with irrelevant facts. In many of these systems, adding a large set of redundant facts will result in a failure of the proof search because of the explosion in search space. Thus, in practice one of the main obstacles of using techniques in automated theorem proving is the lack of scalability and robustness against redundancy. Inspecting the origin of individual axioms with respect to the original application description may help to assess the importance of the axioms and may also help to guide the selection of axioms during the proof search.

Thus, one way to solve the problem of scalability is to transfer the structuring mechanisms provided by the application to control the proof search. Generally speaking, semantic knowledge about the application can often be used to formulate heuristics for the proof process and can reduce the search space dramatically. Rippling [4] as a specific technique to guide inductive proofs is a paradigm of this principle. However, incorporating application knowledge into a theorem proving system requires that it is also able to maintain this knowledge and deduce the consequences for new facts that have been derived with the help of the logical calculus. Tactical theorem proving allows the user to encode application domain knowledge into domain specific tactics and tacticals. However, the user lacks the ability to specify corresponding domain specific datastructures. Annotations [14] may solve this problem in some cases.

In general the embedding of application domains and application specific reasoning into logical formalism is another way to incorporate application knowledge. In the best case the logic is hidden from the user, as she communicates with the system in terms of the application. In interactive systems this results in the need of translation mechanisms from the application to the logic and vice versa because formulas deduced inside the logic formalism have to be represented in terms of the application.

5 Combining Different Proof Techniques

Successful systems are composites like for instance the PVS-system [27]. They combine special purpose procedures (like model checking, arithmetic procedures, computer algebra systems, etc.) to make use of the special knowledge that is incorporated into these procedures. All of them have special restrictions to the kind of problems they can tackle. Within the last years, there is a lot of progress concerning techniques to connect different systems with respect to technical means (see [19] for an overview). Broker and agent-oriented architectures [9,8] have been proposed for providing a flexible infrastructure to combine various provers and computer algebra systems and to exchange problems between them.

While these systems provide the necessary means to build up a distributed network of reasoning tools and support the interoperability between different theorem proving systems, techniques within the individual provers are missing to detect situations in which problems can be successfully delegated to another tool that is available in the network. In order to make effective use of these tools, an occurring problem has to be reformulated such that it fits into the restrictions of the designated special purpose procedure. The reader is referred to [3] for the description of a painful and tedious process of integrating an arithmetic decision procedure into NQTHM [2].

In general, lemma speculation and generalization techniques are required to reformulate the problem to be delegated within the language of the designated

tool. To use model checking techniques, for instance, we have to abstract from infinite states. When applying arithmetic decision procedures, all occurrences of user defined functions have to be replaced by variables and appropriate preconditions on these variables have to be speculated to preserve necessary attributes of the generalized terms.

6 Conclusion

In this paper we proposed the need for a methodology and corresponding techniques and tools to support an evolutionary proof engineering and sketched the issues arising when pursuing this task. While there are already techniques available to solve individual arising problems, there is a need for an integrating methodology and embracing techniques to combine the existing techniques to a successful proof engineering approach.

References

[1] S. Autexier, D. Hutter, T. Mossakowski and A. Schairer. The development graph manager MAYA. In *Proceedings 9th International Conference on Algebraic Methodology And Software Technology, AMAST2002*. Springer-Verlag, LNCS, (2002)

[2] R.S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, (1979)

[3] R.S. Boyer and J S. Moore. *Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic*, Machine Intelligence (11),pp. 83-124, Clarendon Press, Oxford, (1988)

[4] A. Bundy, D. Basin, D. Hutter and A. Ireland. *Rippling: Meta-level Guidance for Mathematical Reasoning*, Cambridge University Press, (2003)

[5] W.M. Farmer. *An Infrastructure for Intertheory Reasoning*. 17th International Conference on Automated Deduction CADE-17, (2000)

[6] W.M. Farmer, J.D. Guttman and F.J. Thayer. *IMPS: An Interactive Mathematical Proof System*. 10thInternational Conference on Automated Deduction CADE-10, 1990

[7] A. Felty, D. Howe: Generalization and Reuse of Tactic Proofs. In: *5th International Conference on Logic Programming and Automated Reasoning, LPAR-94*, Springer-Verlag, LNAI 822, 1994

[8] A. Franke, S. Hess, C. Jung, M. Kohlhase and V. Sorge. Agent-Oriented Integration of Distributed Mathematical Services, Journal of Universal Computer Science, 5(3)156–187, (1999)

[9] F. Giunchiglia, P. Pecchiari and C. Talcott. *Reasoning Theories - Towards an Architecture for Open Mechanized Reasoning Systems* In F. Baader and

K. U. Schulz (eds): Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany), Kluwer Academic Publishers, 157–174, (1996)

[10] F. Giunchiglia, A. Villafiorita and T. Walsh. *Theories of Abstraction*, AI Communications, 10(3-4)167-176, (1997)

[11] F. Giunchiglia and T. Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.

[12] R. Henzinger, R. Jhala, G. Sutre. *Lazy Abstraction*. Principles of Programming Languages, POPL 2002, (2002)

[13] J.R. Hobbs. Granularity. In Arivind Joshi, editor, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1–4, Los Altos, California, 1985. Morgan Kaufmann.

[14] D. Hutter. Annotated reasoning. *Annals of Mathematics and Artificial Intelligence (AMAI). Special Issue on Strategies in Automated Deduction*, 29:183–222, (2000)

[15] D. Hutter, B. Langenstein, G. Rock, J.H. Siekmann, W. Stephan, and R. Vogt. Formal software development in the verification support environment. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(4), (2000)

[16] D. Hutter and A. Schairer. Towards an evolutionary formal software development. In *Proceedings 16th IEEE International Conference on Automated Software Engineering, ASE-2001*, San Diego, USA, IEEE Computer Society, (2001)

[17] D. Hutter and W. Stephan (Eds.). *Mechanizing Mathematical Reasoning, Techniques, Tools, and Applications, Festschrift in honour of Jörg H. Siekmann*. LNAI 2605. Springer-Verlag, 2003.

[18] A. Ireland, A. Bundy: Productive Use of failure in inductive proof. *Journal of Automated Reasoning*, 16(1-2), pp. 79-111, 1996

[19] M. Kerber and M. Kohlhase (eds) *Symbolic Computation and Automated Reasoning – The Calculemus-2000 Symposium*, A K Peters Publishers, USA, (2000)

[20] R. Kling: A paradigm for reasoning by analogy. *Artificial Intelligence*, 2:147–178, 1971.

[21] T. Kolbe, C. Walther: *Proving Theorems by Reuse*, Artificial Intelligence, 116(1-2),pp. 17–66, (2000)

[22] R. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton University Press, (1994)

[23] D. Loveland. *Automated Deduction: Looking ahead*, AI-Magazine. **20**(1) (1999).

[24] W. McCune". *Solution of the Robbins Problem*. Journal of Automated Reasoning, 19(3)263–276, (1997)

[25] T.F. Melham. *Formalizing Abstraction Mechanisms for Hardware Verification in Higher Order Logic.* PhD thesis, Computer Laboratory, University of Cambridge, 1990.

[26] E. Melis: A model of analogy-driven proof-plan construction. In *14th International Joint Conference on Artificial Intelligence*, Morgan Kaufman Publ. pages 182–189, Montreal, 1995.

[27] S. Owre, S. Rajan, J.M. Rushby, N. Shankar and M. K. Srivas *PVS: Combining specification, proof checking, and model checking*, Proceedings of the 18th International Conference on Computer Aided Verification CAV, Springer, LNCS 1102, New Brunswick, NJ, USA, (1996)

[28] D. Plaisted. Abstraction mappings in mechanical theorem proving. In *Proceedings of the Fifth International Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 264–280, Les Arcs, France, 1980. Springer-Verlag.

[29] A. Voronkov. *Algorithms, Datastructures, and Other Issues in Efficient Automated Deduction*, Proccedings of the 1st International Joint Conference on Automated Reasoning, IJACR 2001, Springer, LNAI 2083, (2001)